WHAT IS NOT INVENTED HERE SYNDROME?

Pride is a double-edged sword. The good side of pride gives us confidence in our own abilities and makes us want to strike out on our own to find better solutions. The bad side of pride leads us to ignore the merits of external offerings and contributions.

Both of these aspects of pride can lead us to the “not-invented-here syndrome”.

What is Not Invented Here Syndrome?

Not-invented-here syndrome (NIH) can be defined by a tendency for people and organizations to avoid things that they didn’t create themselves. NIH is often the result of pride that makes an organization believe that they can solve a problem in a better way than pre-existing solutions already do.

Pride isn’t the only contributing factor that inspires NIH. Jealousy can also play a role. Outside of emotional responses, NIH can be the result of financial or legal considerations like in cases of patent infringement or when royalties might be incurred. Various factors can lead to NIH and related tendencies.

This syndrome can be similar to the “let’s reinvent the wheel” syndrome. NIH is found in various facets and industries, such as research fields or in government. There is an inherent tendency to have reduced trust in things that you didn’t have personal involvement in creating. NIH is especially apparent and commonplace in the IT world.

When it comes to IT organizations, NIH syndrome can lead to organizations dedicating internal resources to creating their own technology solutions for problems that have already been solved by other companies. While solving your own problems is a noble pursuit, there's nothing wrong with accepting some help from others. In fact, it's pretty safe to say that the entire technology industry is built upon the successes of our forefathers.

Despite all of this, NIH—like pride—is not inherently good or bad. Much like pride, NIH is a double-edged sword. As such, I'm going to make the case for and against NIH syndrome.

The Case For NIH

Having confidence in your organization's ability to develop strong technology solutions for internal problems is a great sign that your development team is working as intended. Developing your own solutions can provide a myriad of benefits.

Internally developed software is made specifically for the needs of your organization. This means you won't need to adjust software from its original purpose and rig it to meet your specifications. Solutions created by in-house staff will place the needs of the organization at the forefront throughout the entire development process. This means tools created by your organization, for your organization should have everything you need and nothing you don't.

In-house development allows you to have complete control over your technology solutions. If your organization's needs change suddenly, you can immediately put your team to work on making the necessary changes. If you outsourced that software, you're at the mercy of someone else's development team that might already be in the middle of a separate project or have no interest in making the modifications you need.

Developing in-house tools can lead to a new source of income if the tools you make can be sold for use in other organizations. This may not always be the case and attempting to pursue this line can result in bogging down your internal development teams as they try to make more generic solutions as opposed to ones created for the express needs of your organization.

Furthermore, attempting to leverage in-house software as a product can put you in a situation where you now have to devote more resources to that project to make it more suitable for paying customers as their needs shift. The lines begin to blur here when you attempt to transition an in-house solution into a business offering.

The Case Against NIH

While NIH can result in the creation of some excellent products for your organization, it can also lead to a lot of wasted time and resources. The double-edged sword cuts in reverse and can result in your development team being forced to devote excessive time to projects that don't drive much if any value back to your organization. The time your team spends on creating software packages for your organization could be time spent on the products off which your organization actually makes money.

It's important to recognize when you're creating a better solution and when you're really just reinventing the wheel. It turns out, some things (like wheels) are pretty great already. Using internal resources to start from scratch isn't guaranteed to result in the final product being better than what is already available to you.

Using external software solutions allows you to rely on the expertise of other organizations that have a dedicated focus on creating the products they offer. Your development team might be capable of creating accounting software, but if they aren't accounting experts they have a good chance of overlooking some important aspects. Leveraging experts from outside your organization is a great way to enhance your organization's capabilities.

There are tons of tools already out there that have had years of development and refinement put into them. Oftentimes, these tools can be purchased for far less than it would cost to put your own team to work on them. Proprietary tools might be slightly better than those purchased from other companies, but will they be better enough to warrant the development time put into creating and maintaining the tools?

A good general tip for deciding between in-house or outsourced software is to use your internal teams for creating tools with which they have expertise. If you're an accounting software firm, it makes sense that you'd use internally developed accounting software for your own purposes. On the other hand, you probably don't want to bother with creating database software if your company creates audio editing tools.

When it comes to developing in-house versus outsourcing, the decision should be informed by data as opposed to emotion. Leave your pride at home and consider the overall business sense on a case-by-case basis.